

DTIC FILE COPY

①

DISTRIBUTED MANAGEMENT OF
REPLICATED DATA

Final Report

AD-A229 522

DTIC
ELECTE
NOV 30 1990
S D D

October 9, 1984

APPROVED FOR PUBLIC RELEASE
DISTRIBUTION UNLIMITED

90 37 24 151

DO NOT REMOVE
*****00269907**

CONTENTS

	Page
1. Executive Summary	1
1.1 Summary of CCA's Contributions to The SAC C ₃ Experiment	1
1.2 Review of Research and Technology Work Statement	2
2. Project History	7
2.1 The SAC Strategic C ₃ Experiment	7
2.1.1 Technology	8
2.1.2 Demonstration Application	10
2.2 CCA's Role	11
2.2.1 Database Management	11
2.2.2 DACOS User Interface	14
2.2.3 Application Development	14
3. Database Research Results	19
3.1 The Problem	19
3.2 Restoring Divergent Database States	21
3.2.1 Data-Patch	23
3.2.2 Log Transformations	26
3.2.3 Predefined Transaction Interaction Characteristics	28
3.3 A New Approach	31
4. Project References	35

ILLUSTRATIONS

2.1	Levels of Technology	9
3.1	The Problem	20
3.2	Data Patch	22
3.3	Log Transformations	22
3.4	Site Architecture	33

1. Executive Summary

This report describes Computer Corporation of America's (CCA's) contributions to the U. S. Air Force Strategic Air Command (SAC) Command, Control, and Communication (C₃) Experiment. The objective of the Experiment is to demonstrate how emerging communication and software technologies can be integrated to support survivable trans- and post-attack command and control. CCA has provided software research, design, and implementation support in the area of distributed database management technology. This technology will allow systems to provide continuous access to critical information through intelligent, easy-to-use interfaces to replicated and distributed strategic databases.

The first subsection below summarizes CCA's contributions to the Experiment. The second subsection describes how CCA's work addressed each point in the project's Statement of Work. JS/K

1.1 Summary of CCA's Contributions to The SAC C₃ Experiment

The CCA effort in support of the SAC C₃ Experiment has three main thrusts: database management in a hostile environment, the Database Centered Office System (DACOS) user interface model, and application development for a system demonstration.

For this project, CCA's database research focuses on the design of a system that continues to run on replicated databases in spite of communications and host failures. The databases must continue to be available for updates and retrievals, even though access to some of the replicated copies may be prevented. The most difficult obstacle to this goal is reconstitution of a database when communications are restored after a network partition. When communications are received the database must be brought up to date, requiring reconciliation of conflicts that have occurred between local and non-local transactions. Solving this problem required abandoning the traditional standard of serializable execution of transactions in

defining database correctness.

CCA's user interface research was driven by the need for uniform and easy-to-use user interfaces as part of the survivable C₃ system. The SAC C₃ environment requires quick response, integrated interfaces to a variety of systems, and easy-to-use interfaces to complex databases. The resulting interface, DACOS, supports applications needing minimal user training and which are quickly modifiable to deal with unexpected situations. The interface provides an application development system that does not require programming knowledge and that produces applications with great commonality, so that a user trained on the system needs very little training on each new application. DACOS provides automated form/menu selection interfaces for database queries and the SAC application functions that are developed.

CCA's application development involved demonstrating the use of the prototype distributed database technology for representative SAC databases. SAC personnel can access and update these databases without being aware of the geographical location or replication level of the data. CCA worked in conjunction with other contractors to study SAC's requirements, choose a functionally representative subset, and design a prototype database and applications to satisfy those needs.

Details about CCA's role in the SAC C₃ Experiment are provided in Section 2 and 3.

1.2 Review of Research and Technology Work Statement

This section summarizes the work completed to satisfy the Research and Technology Work Statement (SOW). Its organization corresponds to that in [9], addressing the items of Section 4.1.

- o Analyze the SAC C₃ experiment data base requirements. (SOW 4.1.1)

To analyze the SAC C₃ experiment data base requirements, CCA studied the MITRE Corp. reports [13, 14, 15, 21, 24, 25, 26, 27, 28, 29, 30, 31, 32] and participated in a series of meetings and discussions. Topics included: the user interfaces, application programs, and experimental scenarios. The result of these

activities was the decision to make the aircraft recovery function the objective of the system demonstration.

- o Design the SAC C₃ data bases. (SOW 4.1.2)

Based on the MITRE reports, CCA abstracted the underlying data structure and data access requirements needed for the generalized system programs. A Daplex database, to support the aircraft recovery function, was described in [1]. Studies were also conducted to identify the functions that needed to be incorporated in other potential application programs.

- o Evaluate the SDD-1 performance characteristics relevant to the SAC C₃ Experiment. (SOW 4.1.3)

CCA's analysis of SDD-1 consisted of relating its operational requirements to the SAC C₃ operational environment. Incompatibilities in hardware base and operating system environment led to a decision to provide approximate SDD-1 functionality (for the purposes of the experiment) by using a conventional centralized DBMS (Berkeley INGRES) and a multi-site distributor to provide distributed database management functionality. The early results of studies required by SOW item 4.1.4 and a qualitative study of the performance of recovery techniques for network partitioning were used as input in the design of the distributor.

- o Study data base catastrophe and reconstitution. (SOW 4.1.4)

Communications problems like those encountered in the SAC C₃ Experiment would probably cause database catastrophes such as interrupted operation and inconsistent copies of replicated data. CCA studied the effects of intermittent or slow communications on distributed, replicated databases and developed new solutions to the problem of database reconstitution. We reviewed the literature on DBMS operation during communications failures and on database reconstitution methods (see Section 4, and "References"), and investigated the applicability of the proposed solutions to the Experiment. Each of these solutions either prohibited updates to particular data during a communications failure or failed to account sufficiently for real-world actions taken during such failures. Therefore, CCA developed new database reconstitution solutions appropriate to the SAC C₃ Experiment.

Researchers investigated criteria for continuous DBMS operation in the face of induced network partitions and for the correct reconstitution of the replicated databases when communications are restored (see Section 3.1). Two solutions to the problem were developed: data-patch and log transformations (see Section 3.2). This work led to the formulation of a general architecture for handling more complex types of communications failures (see Section 3.3).

- o Design a Data Base Administration (DBA) reconstitution workbench. (SOW 4.1.5)

CCA's development of the data-patch and log transformations methods for database reconstitution after network partitions form the basis for a set of useful DBA tools. These tools (primarily in the form of guidelines) serve the functions of application design, analysis of database damage due to communications problems, and support of database reconstitution techniques. Both data-patch and log transformations rely on the DBA to characterize data types or user transactions in terms of certain specific properties. The efficiency of the patch and log approaches in different situations has affected the design guidelines that are being incorporated into DBA tools. These tools are embodied in a module called the Checker (described in Section 3).

- o Perform investigation of a Data Base Centered Office System (DACOS) to permit uniform interfaces to the SAC command and control environment. (SOW 4.1.6)

During 1982 and 1983, CCA completed a design of DACOS. It was documented in four reports [5, 12, 16, 17]. A limited version of the system has been implemented to support SOW items 4.1.7 and 4.1.8.

- o Perform data base structure, workbench tools definition and application program design. (SOW 4.1.7)

CCA specified an INGRES format for the aircraft recovery database, and designed the unclassified applications and ad hoc query facility. (See also SOW 4.1.2)

- o Implementation of the data base and selected workbench and application programs. (SOW 4.1.8)

The unclassified portion of the database, which included actual contents of War Readiness Spares Kits

(WRSK), and a simulated personnel roster were converted from source documents and loaded into an INGRES database. Effort was concentrated on implementing the unclassified applications and a prototype of the ad hoc query facility. These were demonstrated in November 1983.

- o Design and coordinate SAC C₃ data base experiments. (SOW 4.1.9)

CCA provided input to and worked with MITRE and SRI on the production of [1, 2, 3]. Also, under the scope of this item, CCA designed and worked on the development of a secure computing facility to accommodate the classified nature of the applications. In addition to coordinating security arrangements with SRI, RADC, and DARPA to develop secure communications, CCA worked to develop an in-house secure computing facility by acquiring a VAX 11-730 for use on this contract.



Accession For	
NTIS CRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

2. Project History

This section ties together the technical reports in the Appendices by giving a history of the events which influenced the evolution of the project. As with any research project, CCA adapted the direction of our research and development efforts to meet the evolving needs of the user community.

2.1 The SAC Strategic C₃ Experiment

SAC has a global mission that requires the ability to continue command and control operations during and after receiving a debilitating attack. In recognition of this need, SAC deployed the Airborne Command Post (ABNCP) to provide mobile command and control in the event that fixed facilities are damaged or destroyed. In a protracted conflict, however, additional means would be needed to identify, reconstitute, and employ surviving assets.

One concept that SAC is exploring is a ground-based transportable command center, the Headquarters Emergency Relocation Team (HERT). Each HERT maintains a replicated SAC command capability and is an element in a coordinated network of high survivability. The success of this concept depends on the availability of technology to both military and civilian personnel that can provide a distributed, flexible network of computers, each element of which is capable of maintaining strategic information about the status of personnel and equipment. The computer network also will be the basis for communication between team members at different geographic locations.

In 1979, DARPA proposed a joint experiment to test technologies developed in their labs [DARPA 1982]. The experiment would use ARPANET, packet radios, and network security and database technologies to provide an architecture capable of supporting trans- and post-attack command, control, and communications requirements. After that proposal, a Memorandum of Understanding (MOU) for a Strategic C₃ Experiment was prepared between the Defense Communications Agency World Wide Military Command Control System (WWMCCS) Systems Engineer (DCA/WSE), DARPA, and SAC. As

specified in the MOU, some of the desired features of the proposed experiment are to:

- o Demonstrate and evaluate the survivability of a multi-node computer communications network;
- o Demonstrate remote access to databases from surviving elements -- the HERT and ABNCP C₃ elements;
- o Demonstrate rapid reconstitution of C₃ networks following simulated damage to network structure.

2.1.1 Technology

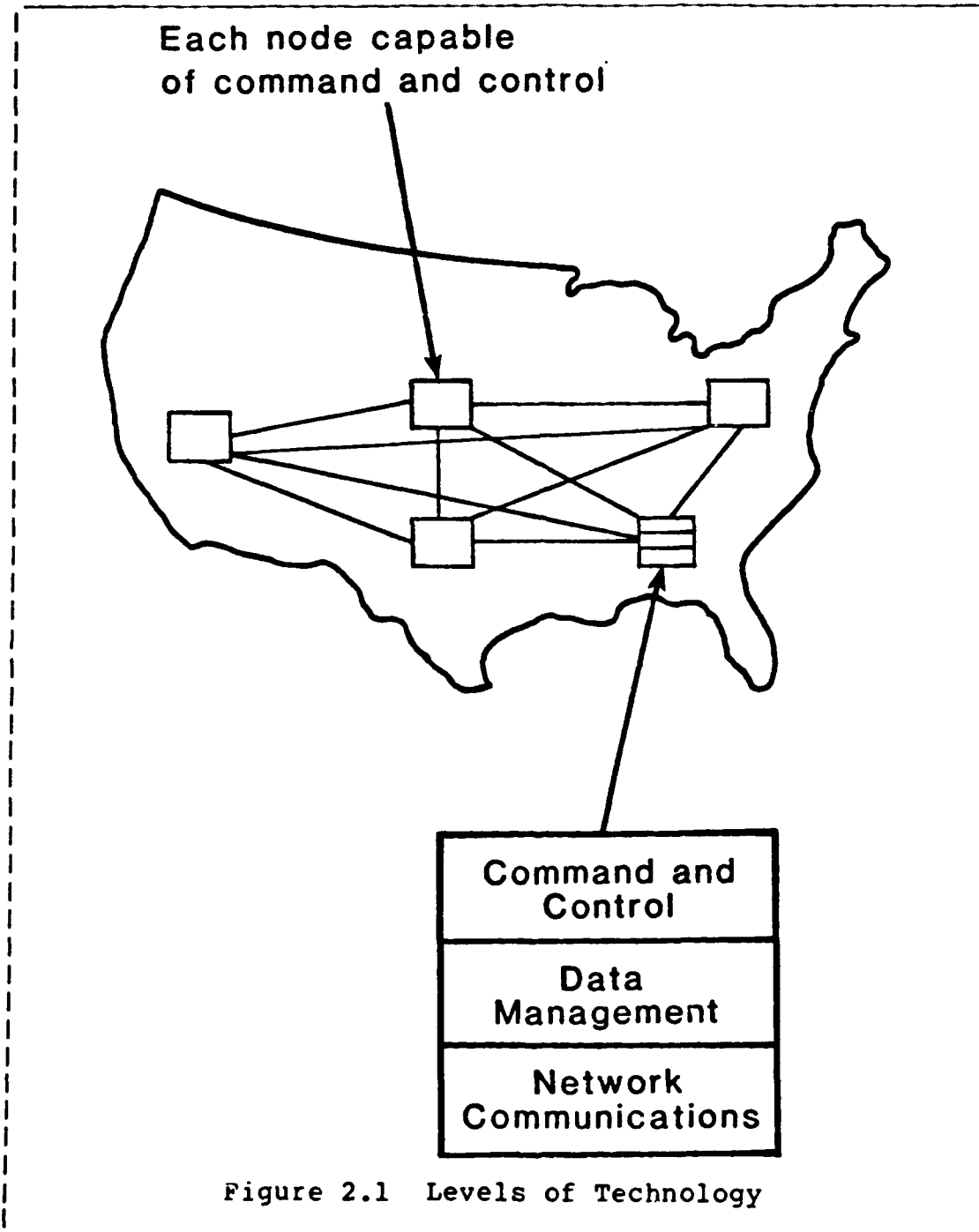
Based on the objective of developing and demonstrating a network that will allow command and control from any point in the network, a successful prototype system must show feasibility at three levels, as shown in Figure 2.1. The levels represent the three types of system interactions that are required of the applied technology:

- o User interfaces; command and control capabilities.
- o Databases: data management considerations,
- o Communications: network design and maintenance,

To demonstrate feasibility at the communications level, the experiment must show that it is possible to implement a secure packet switched radio network that can operate in the presence of damaged nodes. Nodes might become temporarily inaccessible because of adverse electromagnetic conditions and at a later time might re-enter the network. This demonstration is described in Section 4.1 of the Strategic C₃ Experiment Program Plan.

To demonstrate feasibility at the databases level, the experiment must show that it is possible to design and create a database of strategic information using a database management system (DBMS). Such a DBMS must:

- o Permit the distributed storing of data throughout the network;
- o Support replication of strategic information at many sites in the network;



- o Tolerate failure and possible subsequent recovery of computer systems and lines of communication, including possible network partitions, while supporting use of all operational resources;
- o Support concurrent access by multiple users.

Demonstration of these capabilities depends, in part, on the communications facilities demonstrated at the first level. The databases demonstration is described in Section 4.2 of the Strategic C₃ Experiment Program Plan and is supported by the effort described in this report.

The third level is the development of user interfaces for command and control. The experiment must demonstrate that it is possible to generate command and control systems at each node of the distributed network. These systems must be both powerful yet useable by personnel having minimal training and experience. Capabilities such as extended electronic mail and access to distributed databases will be demonstrated through the use of a forms-oriented office automation interface. These capabilities will depend on the distributed database capability of the second level and the communications capability of the first level. The user interface is described in Section 4.3 of the Strategic C₃ Experiment Program Plan.

2.1.2 Demonstration Application

For the purpose of this experiment, the three levels will be brought together to support the Aircraft recovery function. In this demonstration, the SAC C₃ experimental system is used to aid in directing returning aircraft to appropriate bases. A returning aircraft should be directed to a surviving base that can support the repairing, refueling, and re-arming required to ready the aircraft for future missions.

Three SAC agencies (Operations, Logistics, and Combat Plans) will use the Aircraft recovery function. Each agency has an area of responsibility that determines the types of data and the application programs that are used by agency personnel. The distributed database supporting the aircraft recovery function is shared by these agencies. The use of a single integrated database allows data to be shared in a controlled manner, without having to explicitly pass it from one agency to another.

It is important that the demonstration prototype for this technology be as realistic as possible, to test the resilience of the system to the type of changes that might be common in operational systems. The system that the aircraft recovery function replaces is a manual ("grease pencil") database. Exact needs of users for an automated system are not well understood and cannot be determined with any degree of certainty from current procedures. (The appropriate refinements may be established when a prototype system is in place.) Thus, the system produced must be adaptable to meet evolving requirements easily and rapidly as they are defined.

2.2 CCA's Role

The three main areas of CCA's work in support of the SAC C₃ Experiment were summarized in Section 1: database management in a hostile environment, the DACOS user interface model, and application development for a system demonstration. The results of these activities are addressed in reports that have been filed as deliverables of this contract and others that are included as appendices to this report. What follows is a history of the events that shaped the evolution of the CCA project.

2.2.1 Database Management

The database research task consisted of designing and demonstrating new techniques to provide database robustness in a hostile environment. The proposed architecture is a fully replicated and geographically distributed DBMS that can continue to operate in the presence of arbitrary combinations of site and communication-link failures. This task was completed essentially as planned in spite of changes in choice of the underlying DBMS twice in the course of the project.

The first part of our task was to develop the means of maintaining the distributed and replicated databases regardless of communications failures and recoveries. Our goals were

- o To allow any user who could establish a connection to any functioning database site;

- o To allow full access to and updating of the database;
- o To have the database system automatically resolve any conflicting updates when network connectivity was restored.

From January through November of 1982, we developed two viable approaches to the network partition recovery problem: Log Transformations and Data Patch. These approaches are documented in [6, 8, 10, 18] as well as in Section 3 of this report.

During 1983, we developed a third approach that deals with slow as well as unreliable communications. It builds on a combination of the Log Transformations and Data Patch approaches. This third approach is necessary because the distributed database algorithms that normally maintain serializability of transaction execution are subject to severe performance degradation when communications are slow. To get around this issue, we used a weaker standard of database correctness to resolve these potential performance problems. This approach is outlined in [7] and in Section 3 of this document.

The second part of the database research task was designing a prototype system for demonstrating the research results and for use in the SAC C₂ Testbed. The original plan, as outlined in our contract proposal, was to implement a Daplex front-end to CCA's SDD-1 distributed database manager and to build database reconstitution protocols into that system. SDD-1 can run on a network of DEC10s and DEC20s. The subsequent choice of VAX/UNIX* as the environment for testbed software made the use of SDD-1 difficult. We briefly considered keeping the database under SDD-1 and accessing it through a network from the VAXs of the testbed. In February 1982, we decided that a more reasonable approach was to build the database reconstitution algorithms into the LDM/DDM DBMS that was then being implemented by CCA. This DBMS already used Daplex as a query language, so it would not need a translator. Also, it was expected to be running on the VAX/UNIX testbed environment in time for the total system demonstration.

*UNIX is a trademark of AT&T Bell Laboratories.

In January 1983, the research had progressed to the point where we were ready to begin detailed design of the prototype. Because the LDM/DDM implementation schedule had slipped (for lack of an Ada** compiler), it was not available at this phase. We then determined to implement the first prototype system on top of the INGRES relational DBMS that is provided in the standard UNIX operating system package. We left open the option of converting to the LDM/DDM under a follow-on effort when that system became available. As the LDM/DDM schedule continued to slip, that option became increasingly remote.

In March 1983, we proposed a phased implementation of the distributed database system. In the final implementation, the distribution software will have two major components:

- o A Distributor, which is responsible for reliable broadcasting of update notifications to all sites;
- o A Checker, which is responsible for generating the correct database state even when update notifications arrive out of order.

In the first implementation phase, the applications would be broken into interactive and update "steps". The update steps of the applications would carry out the functions of the Checker in the eventual architecture. A single site distributor would duplicate the communications protocols within a single machine and allow debugging of the applications, which could then run without change when the full distributor became available. Finally, the Checker module would replace the application update steps.

In July 1983, we released a specification of the program interfaces to the Distributor [9]. Also at that time, we implemented a "dummy distributor" that emulated the actual interfaces and thus allowed program modules to be debugged in a single site environment.

Implementation of the single site Distributor was first held up by delays in receiving the 4.2BSD release of UNIX. (That release contained an Interprocess Communications Facility (IPC), without which the Distributor could not be implemented efficiently.) Following installation,

**Ada is a trademark of the Ada Joint Program Office, U. S. Government.

we implemented the single site distributor but were unable to demonstrate it owing to a series of difficulties with the IPC that were not resolved until after the contract was completed.

2.2.2 DACOS User Interface

DACOS is a design task aimed at creating an office system whose components communicate primarily by updating a distributed database. Major components of this system include:

- o Dialog Manager: creates a uniform form-based user interface shared by all of the applications that are defined within DACOS;
- o Form Procedure Definition Utility: allows applications to be created by completing the appropriate forms;
- o Form Procedure Interpreter: executes applications and uses the Dialog Manager;
- o Ad-hoc query facility: is both an illustrative application and a useful tool in maintenance of the system.

DACOS design took place between May 1982 and May 1983. Four documents [5, 12, 16, 17] provide details of the user interface. This design was completed as planned. Appropriate implementation of the DACOS system is planned for the follow on effort for use in creating the applications for the system demonstration.

2.2.3 Application Development

The application development task of this project underwent large changes, all ultimately driven by the need for secure facilities for classified system development. There were no plans for classified development in the original proposal. Secure facilities did not ultimately become available until after the completion of this contract, and as a result the programs written were general and illustrative only.

The first phase of this project involved meeting with SAC and other contractors to discuss requirements and options. In March 1982, we reviewed the MITRE

requirements document [13, 14, 15]. In April, we participated in a joint decision to make the aircraft recovery function the focus of the system demonstration. In May 1982, we released a draft document that described a Daplex database to support the aircraft recovery function. After the document was reviewed and modified, it was released in final form in August [1]. At that time, it was expected that the DBMS used to support the experiment would be CCA's LDM, a Daplex-based database manager.

The document issued in August 1982 was the last unclassified document to be released as part of the design effort. It was based largely on our discussions with SAC and CCA research team ideas, since we could not access the classified documents describing the existing system. The original intent of the Experiment community was to perform an unclassified demonstration. Even when we discovered that most of the data and procedures related to the Aircraft recovery function were classified, we still hoped that we could use an unclassified subset of the information. It was not until December 1982 that the decision was made to secure the testbed and to demonstrate at least some classified applications.

Between July 1982 and January 1983, the application development task grew as related functions were defined and incorporated. Important factors in deciding which capabilities should be included in the demonstration system were

- o Need for integrated set, so that all required data would be available;
- o Need to insure that all the SAC functional areas were represented, so that the demonstration would be meaningful for all participants.

By January 1983, the set of C_3 program modules had grown to approximately 50. There were 30 data entry "reports" and 20 data retrieval "displays." These were to be implemented similar to SAC's existing system, which was partially automated and partially manual. The specifications for that system were classified, and we were unable to obtain them until January 1983.

Between January and April 1983, we developed a new document [2] that defined the database needed to support the enlarged set of programs. The decision had been made at that point to use the INGRES relational DBMS instead of CCA's LDM, because of problems getting an Ada compiler

that would run on UNIX. Because of the conceptual advantages of Daplex, we decided to continue using the Daplex data model to describe the database and to map the Daplex description down to a description in terms of relations.

It was known by early 1983 that at least some of the applications would have to be classified, and therefore that we would need access to secure development facilities to create the database and the applications. We expected to gain such access via an encrypted network link to SRI in July 1983. To keep the amount of classified development to a minimum and to allow the development to begin before secure facilities were available, we designed a report interpreter that would allow the 30 data entry reports to be implemented as parameter files to an unclassified forms package.

In June 1983, it was clear that the July deadline for secure facilities could not be met due to difficulties in obtaining secure communications hardware. The new projected date at that time was October. Furthermore, all but two of the reports had been determined to be classified. Since SRI's secure facilities were still expected to be available on time, a decision was made to have CCA continue to work on the report interpreter and the two unclassified applications (NUDET and GLASSEYE), with a third classified application (ENTRY) to be implemented when secure facilities became available.

In July 1983, MITRE released a new document restructuring the applications so that data input and retrieval would be done through a single form [25]. The expanded versions of the NUDET and GLASSEYE reports in this new document were classified, allowing for no unclassified applications. We designed enhancements to the report interpreter to support the updated applications. The new system contained much of the functionality of the Dialog Manager that we planned as part of DACOS for the follow-on effort. We continued to develop based on the older, unclassified NUDET and GLASSEYE reports as demonstration prototypes.

The enhanced report interpreter took longer to debug than planned. Certain problems came to light only when it was run on the 4.2 version of UNIX at SRI, and there were communications problems that hampered debugging of the system at SRI. The program interface to the report interpreter is documented in [19]. This document was reissued periodically and traces the history of the enhancements.

By November 1983, we had implemented the older unclassified versions of the NUDET and GLASSEYE reports and developed a prototype of the ad hoc query facility as a part of DACOS. We demonstrated them at the November experimenters' meeting. Secure facilities projections had slipped again. (Classified development facilities did not finally become available until May 1984 at SRI and October 1984 at CCA.)

In February 1984, MITRE provided us with documentation for a newly-defined unclassified PERSONNEL application. That application required still more capabilities in the report interpreter. While the report interpreter was originally intended to be used as a stopgap until the DACOS facility could be developed, two factors led us to decide to enhance to report interpreter to the degree necessary to support all the SAC applications except the ad hoc query facility. These two factors were delays in the start of the follow on contract (funding the development of DACOS), and the pressure imposed by a March 1985 deadline for demonstration of the entire system.

The effort to enhance the report interpreter was mapped out in March 1984 -- in the final days of the contract -- to extend to September 1984 under the follow-on contract. The report interpreter, renamed DACOS I, would support the same end-user interface as the full DACOS system but would not provide the application development tools. Program modules developed before the full DACOS system became available would take much longer to implement but would be compatible from the user's viewpoint.

3. Database Research Results

This section provides a highlighted summary of the research results in the area of maintaining replicated data in the presence of network partitions. Discussions that follow amplify the documents contained in the enclosed appendices.

3.1 The Problem

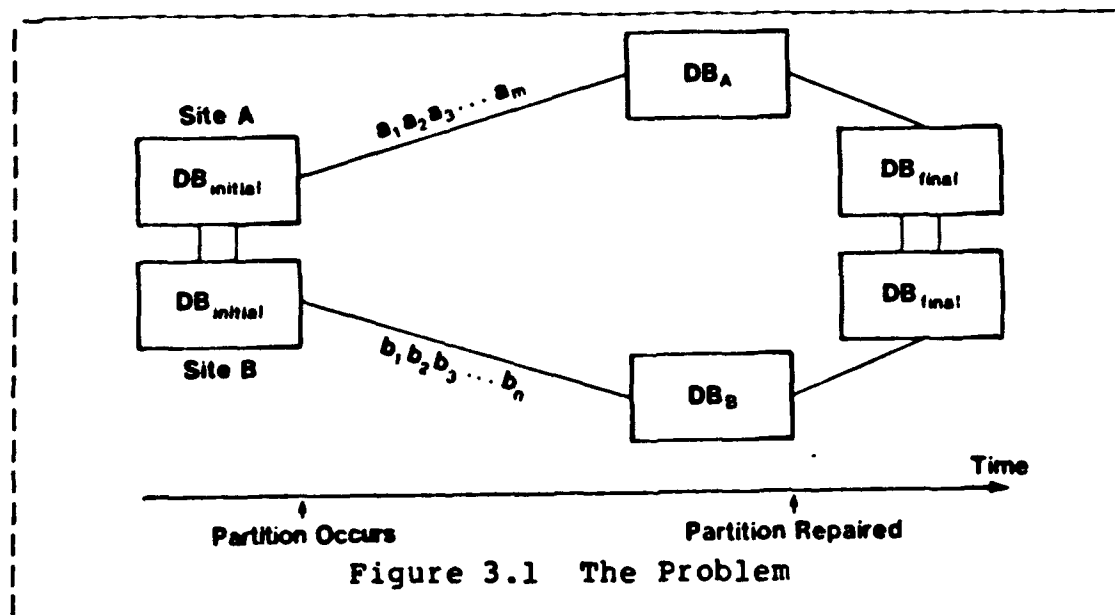
Replicated databases (databases in which copies of critical data are stored at various different sites) are typically useful for applications in banking, the airlines, and military command-and-control that depend on the ability to retrieve data at all times. A replicated database must satisfy two main requirements: availability and survivability. Data in a replicated database is more available because many operations can be handled locally and, therefore, more efficiently. Also, site failures are less catastrophic -- even if the local site is down, another operating site is likely to have the necessary data. Replicated databases increase data survivability, for if data at one site should be destroyed the data may survive at other sites. However, these advantages also introduce concerns which are of lesser impact in centralization database systems.

Specifically replicated databases are more susceptible to communication failures and data inconsistencies introduced by local-mode updating. Copies of data in replicated databases should be consistent. When data is updated at one site, that site must maintain consistency by broadcasting the updates to all the other sites. Communications failures, therefore, have a drastic effect on the consistency of replicated databases. Although it is possible to handle communications failures by blocking updates at some sites until communications are restored [4, 11, 20, 22, 23], such a solution ignores some of the reasons for using replicated databases in the first place. As part of the contract, we developed techniques to allow local updates during communications failures and then to restore consistency of the other replicated copies when communications are restored. This research will continue

under the follow-on contract.

We approached the problem by considering first the special case of failure-induced network partitions, situations in which some set of sites cannot communicate with any other sites. We then began work on extending these ideas to handle other types of communications failures including intermittent recovery periods--which do not last long enough for a total reconciliation--and slow communications--where messages are delivered but normal distributed concurrency control is too slow to be usable.

The problem caused by an induced network partition is illustrated in Figure 3.1. Before the partition, sites A and B have mutually consistent copies of data. During the partition, site A processes transactions a_1, \dots, a_m and site B processes transactions b_1, \dots, b_n . When the partition is removed, the sites are left with divergent copies of the data. The goal is to devise an automatic procedure for restoring the consistency of the copies.



Our first task was to define correctness criteria for this situation. Mutual consistency of the copies of data is essential, but not sufficient. Rewriting one copy to match the other (or even changing all data to a constant) suffices to ensure mutual consistency, but the correctness of the system is sacrificed. Under this approach, some or all updates would simply be ignored. The criteria we wanted restores consistency by taking into account all updates made at all sites during the partition.

When restoring consistency, it is also important to consider the real-world events that occurred during the partition. Although it is possible to change database values that restore data consistency, they may reflect inaccurately actions in the real world that actually cannot be altered. Therefore it is not sufficient to rerun all transactions at all sites. For example, once an aircraft has left on a sortie, it is not possible to change the fact that the aircraft has physically moved from its home base. When updates are allowed during a partition, the serializability of transactions is compromised (that is, running transactions during a partition is likely to produce different results than running the same transactions at a time when the sites can communicate). For many applications, it is preferable to compensate for actions taken during a partition than to prohibit transactions during the partition. If an aircraft has left on a sortie that was cancelled at another site during the partition, it becomes necessary to perform a compensating action by redirecting the aircraft.

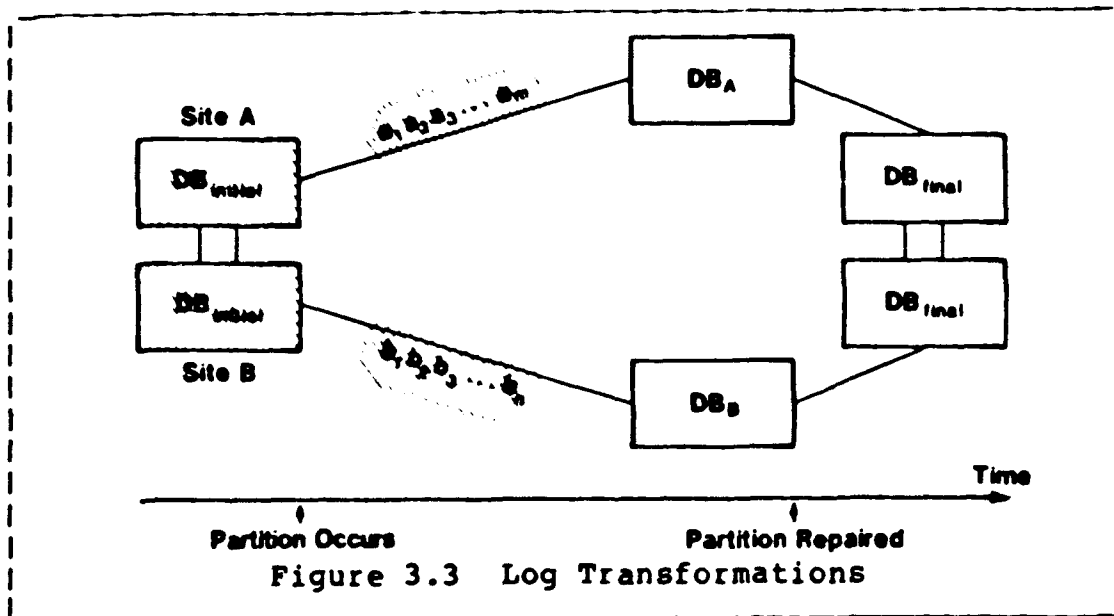
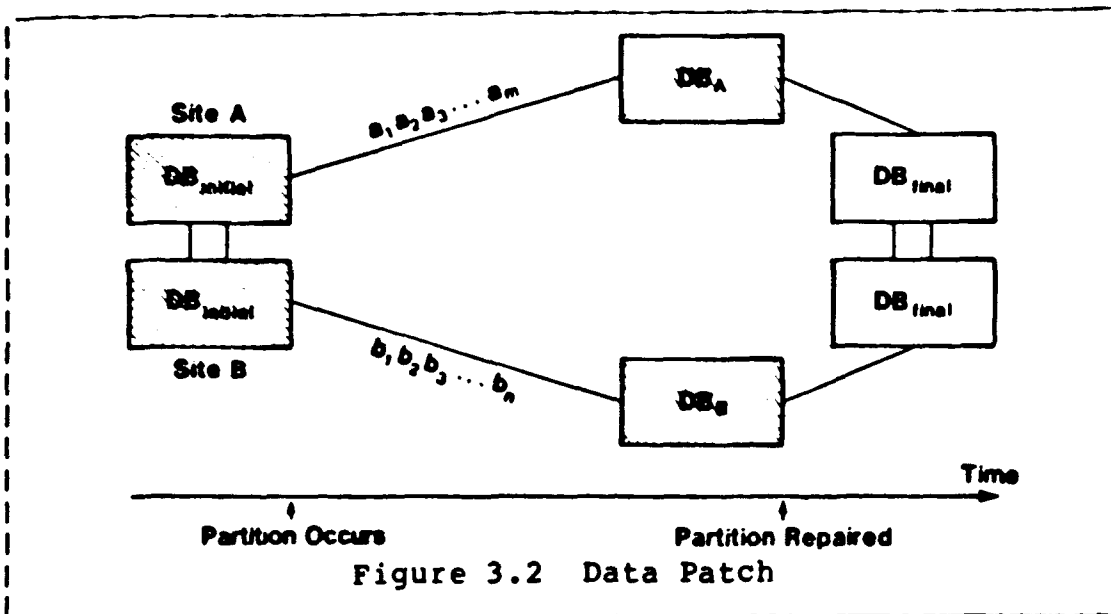
To sum up, after a partition the procedure to restore the databases must:

- o Make the replicated copies mutually consistent;
- o Take into account all transactions executed during the partition;
- o Reflect real-world actions and any necessary compensating actions.

3.2 Restoring Divergent Database States

Our research into restoring the divergent copies of the database that exist after a failure-induced network partition followed two fairly independent approaches. The first approach, data-patch, restores copies of the database by examining the data values that exist at the different sites. The information used in data-patch is highlighted in Figure 3.2.

The second approach, log transformations, is fundamentally different from data-patch in that instead of relying on data values for integration, it uses logs of transactions executed at the different sites, as illustrated in Figure 3.3.



Both approaches have their own advantages and disadvantages; each works well for particular types and frequencies of applications. The performance trade-offs involved in each approach and methods for combining the best aspects of both in a hybrid system, will be studied further under the follow-on contract. In the rest of this section, we will briefly discuss the two approaches.

3.2.1 Data-Patch

The "data-patch" approach to the problem of restoring consistency is based on the data values before the partition and the data values at the different sites after the partition, as illustrated above. Data-patch examines the initial database values and the post-partition values at all sites and constructs a merged database state that reflects all updates at all sites. It then updates all replicated copies to match the merged state. The different data values are merged according to predefined rules, and the resulting data values compose the new consistent copies [10].

Data-patch is a tool used by the database administrator (DBA) to produce a program that integrates the different data values at different sites after a partition. The DBA must first define rules for each type of data in the database definition. Examples of data-patch rules are

- o Computed attributes. The DBA gives a formula for computing each attribute that may be totally derived from other attributes in the schema (for example, average salary, number of account balances over \$100,000). These computed attributes may be either physically stored in the database or computed anew each time they are referenced. For the purpose of integration, the DBA can think of these attributes as being computed each time they are referenced.
- o Data insertion rules. For each data type in the schema, the DBA gives a rule that specifies what is to be done with a data item of that type that was inserted during the partition at one site but not at another. The rules include (but are not limited to):
 - Keep the data item and add it to the databases at other sites;
 - Remove the data item -- delete it from any site(s) where it was inserted;
 - Notify the DBA that an unanticipated situation has occurred and go on to the next data item. A program written by the DBA may also be called. The program takes as a parameter the data item that was inserted, and can access database values at all sites, time stamps associated with those values, and the transaction execution logs at all sites. When this program completes, either the inserted data item will be in the databases at all sites or it

will be in none.

- o Data item integration rules. These rules specify how two data items, D_1 at site 1 and D_2 at site 2, of the same data type and with the same unique identifier, are to be integrated. The result is a single data item D (with the same unique identifier as D_1 and D_2) that is stored in the restored database. Rules are² given for integrating each attribute (except unique identifiers and computed attributes) of the data type. The choices for an attribute (A) include: use the latest value of A ; use the value of A written at a certain site; notify the DBA if the values of A are not identical and go on to the next data item; if the values of A differ, store a constant (useful for storing a null value when a good integrated value cannot be found); notify the DBA if the values of A are not identical and halt. As before, the DBA may also write a program to integrate the values.

Another data item integration rule is the "arithmetic" rule. The integrated value is set to the sum of the values in the different databases minus the initial value before the partition. For instance, suppose that transactions which report additions of boxes to a warehouse are run during a partition. Before the partition, the number of boxes is reported to be 10. When the partition ends, site 1 reports 15 boxes and site 2 reports 12. Then the actual number of boxes in the warehouse should be: $15+12-10=17$.

For brevity, we do not discuss other rules like MAXIMUM and LOGICAL OR that data-patch supports.

- o Integrity constraints and corrective actions. The DBA specifies each constraint that must be satisfied by the restored database. For each constraint, the DBA also provides a corrective program to be called when a violation is observed. For example, in a banking application there might be a constraint that the balance of each account should be greater than zero. The corrective program could set the status of the account to "overdrawn" and charge the customer a fine. Note that not all constraints may require corrective actions (for example, flight overbookings).
- o User notification rules. For each data type, the DBA provides a list of users who should be notified of the integration of values of that type. The user names in the list may be expressions to be evaluated when the

integration occurs. The DBA may also specify whether the messages for a given user should be batched or not. In the banking example, even if a network partition allowed several overdrafts on one account, a single notice is sufficient.

Data-patch has several strengths and weaknesses; its efficiency and usefulness depend on the particular application. In some applications, it can minimize network traffic, since many changes to a data item are coalesced and only one value for each data item at each site need be sent to other sites. When a transaction updates many types of data items, however, that single transaction is responsible for heavy network traffic.

Because data-patch relies on the data values that exist at all sites at the end of a partition, it cannot handle more complex communications failures, such as partitions during integration, or overlapping partitions among more than two sites.

Data-patch allows ad hoc updates; however these may need to be restricted so that the predefined integration rules remain appropriate. Errors may be introduced if integration rules are not kept up to date as transactions are added. Data integration is efficient when the transaction logs do not need to be interrogated, but using the logs may make the integration much more complex. Similarly, if compensating actions are based only on the database values that exist at merge time, they are easy to define and relatively efficient to execute. If, however, the compensating actions are determined in part by the history of which transactions caused certain situations, they are much harder to define and execute.

The details of the Data Patch approach can be found in references [6] and [10], included in the appendices.

The approach described in the next section focuses on techniques that use transaction logs to restore consistency.

3.2.2 Log Transformations

The log transformations approach to restoring divergent databases relies on the logs of transactions executed at partitioned sites. Logs are merged to produce the restored database states at each site. We define the restored database state that should exist at each site after the merge process in terms of the database state that existed before the partition and a target log. The result of executing the target log in the state that existed before the partition is the restored database state; this state will exist at all sites after the merge process is completed.

Rather than having each site revert to the state that existed before the partition and then run the target log, each site begins with its current state and runs a different merge log. A merge log may prescribe rolling-back transactions, rerunning local transactions, running non-local transactions, and running compensating transactions. The thrust of the log transformations research is to find efficient merge logs that are correct; that is, such that the effect of running each site's merge log in its current state is the same as running the target log in the database state that existed before the partition. In this way, the correctness of the restored copies of the database state is maintained.

Using this definition to restate the criterion of mutual consistency, then, our goal is to execute correct merge logs at each site. The other two criteria for the restored database states are

- o To take into account all transactions executed during the partition;
- o To account for real-world actions and compensating actions.

These criteria come into play in defining the target log.

The target log definition relies heavily on application-dependent semantics. The target log is generated automatically, based on:

- o The logs of transactions executed at the individual sites during the partition (partition logs);
- o A system-wide algorithm for agreeing on the order of transactions executed during a partition;

- o Predefined rules for handling compensating transactions.

The target log contains all transactions executed during the partition, ordered according to the system-wide algorithm, and any appropriate compensating transactions, positioned in the log according to policies defined earlier by the database administrator (DBA).

Our strategy for merging divergent database states depends on some knowledge of the application. There are two key areas in which this knowledge is essential: in determining the target log, and in defining the semantic properties that are used to transform the merge log. We discuss these two areas in turn.

To automatically generate the target log, the DBA must define the compensating transactions used to handle undesirable situations. In essence, these transactions test for inconsistencies and take appropriate actions that may range from notifying a particular user to invoking a number of other transactions. The DBA must define both the conditions which are to be tested and the actions which must be taken. The conditions are expressed as predicates over the database state (for example, no aircraft may be directed to fly to an inoperative base). The DBA must also specify the position of compensating transactions in the target log. This decision is typically a matter of policy. For example, in merging the partition logs the system may test for aircraft assigned to inoperative bases after every new sortie or base-status report, after all transactions for a given period, or after all transactions executed during the partition.

Another policy which affects the definition of the target log is the use of conditional statements in transactions. The original decisions to run a particular transaction with particular arguments may depend on data read from the database. For example, an aircraft may be directed to a certain base as a result of reading that base's status from the local database. Such decisions could be incorporated in the transaction itself, but doing so complicates the definition of the roll-back transaction and makes it possible for the transaction to perform very different actions when executed as part of the merge log than it did as part of the partition log. Suppose, for example, that a transaction assigns an aircraft to the nearest operative base. If transactions executed at different sites during the partition change the statuses of various bases, these changes would be reflected in the merge log, and the original transaction may well assign

the aircraft to a different base. Therefore, it is useful to think of the transactions as consisting of two parts: a step which reads the database and determines the action to be performed, and a step which actually reports the action. Thus, in this example there would be one step which determines the closest operative base and another which would assign the aircraft to that base. The second step would be unchanged in the merge log; any inconsistencies resulting from changes to a base's status could be handled by compensating transactions.

Once the target log is defined, application-dependent semantics are used to define a correct and efficient merge log. The log transformation approach assumes that all transactions are predefined (that is, no ad hoc update transactions are allowed). It relies on the DBA to specify two types of application-dependent data: rollback transactions and transaction interaction characteristics.

Rollback transactions must be defined for each transaction. A rollback transaction may undo the effects of its associated transaction either by performing an inverse action (for example, subtraction and addition) or simply by restoring the values that existed before the transaction was executed of all data written by the transaction. In some cases, the set of transactions that can be invoked by the user will include rollbacks for some of its members, for example, one transaction records deposits and another withdrawals or a transaction that adds a (possibly negative) value may act as its own rollback; the DBA must make the relationship explicit.

3.2.3 Predefined Transaction Interaction Characteristics

Predefined transaction interaction characteristics are used to generate efficient merge logs. These characteristics, key to the log transformation techniques described below, describe the way in which the order of transactions within a log may be changed. For example, one characteristic is that two transactions commute, (either may be executed before the other) and the same database state will result when both are finished. These characteristics are highly dependent on the application, and they must be specified when transactions are defined. The DBA must specify which pairs of transactions commute and which transactions overwrite the results of which other transactions.

Once the target log is defined, it is trivial to define a correct merge log for each site. Suppose we have a target log (T) and suppose that D is the database state that existed before the partition. Then, for each site (S) we can roll back each transaction executed at S during the partition, thus returning to the state D. Then we can run T. This log consisting of the rollback transactions followed by the target log is called the initial merge log for S.

The interesting problem is to define a correct and efficient merge log for each site. To do this, we begin with the initial merge log, exploit predefined semantic properties of the transactions, and generate a shorter, and so more efficiently executed, equivalent final merge log. The merge log at each site can be transformed independently of the others.

Merge logs contain two categories of transactions: rollback transactions for transactions that appeared in the site's partition log, and forward transactions that appear in the target log.

Our goal in transforming the initial merge log is to delete transactions that have no effect on the final state produced. For example, if the target log contains a number of transactions that report the same aircraft's location, we can delete all but the latest transaction and be left with an equivalent log (one which results in the same database values). We characterize pairs of transactions in a merge log according to their semantics in order to determine which transactions safely may be deleted. We consider every pair of transactions in a log to be either overwriting, commutative, or conflicting. Essentially, a pair of transactions, T_i and T_j , is overwriting if running T_j alone produces the same result as running T_i followed by T_j ; the pair of transactions is commutative if running the two transactions in either order produces the same result. The pair is in conflict if it is neither overwriting nor commutative. The formal definitions are a little more complex and cover some subtle interactions among transactions.

To determine whether a particular pair of transactions are overwriting, commutative, or conflicting, we of course must analyze the actual values of the transactions' arguments and the values read by the transactions. This analysis may be very detailed, or it may be nothing more than a "worst case" based on the transactions' types and their roles in the merge log. The real question is how much analysis will be done at the time of the database

merge, as opposed to the basic characterization done when a transaction type is defined. More study is needed to determine when the cost of analyzing transactions offsets the losses involved in using the "worst case" analysis. In any event, once pairs of transactions are characterized, the characterizations can be used to delete transactions from the initial merge log.

Based on these characterizations, successive deletion of transactions eventually yields the final merge log. There are two situations in which we delete transactions. The first is when transactions in an overwriting pair are adjacent; we can delete the overwritten transaction and produce a correct merge log. The second situation is a bit more complicated. Under certain conditions, we can show that when a rollback transaction is immediately followed by its matching forward transaction, the two transactions taken together do not affect the database state: both can be deleted. When a log meets the conditions necessary for this double deletion to be correct, we say it is regular.

Commutability is the key to both kinds of deletions. Given a regular merge log L , we use commutability to find (whenever possible) an equivalent log L' in which either transactions in an overwriting pair are adjacent, or in which a rollback and its matching forward transaction are adjacent. Using the definition of an overwriting pair or the definition of regularity allows us to delete either the overwritten transaction or the rollback and its forward transaction, respectively. This deletion produces a new merge log L'' . We prove that, as long as some minor restrictions are respected, L'' is both correct and regular; therefore the process of deleting transactions may continue.

We use a graph to describe the properties that hold between pairs of transactions in a merge log. Graph transformations that delete parts of the graph mimic log transformations that delete transactions in a merge log.

The efficiency and usefulness of the log transformation approach, like data-patch, depends on the particular application. It reduces network traffic when one transaction updates many types of data items; however, transactions that change small numbers of data items cause more network traffic under the log transformation approach than under data-patch.

Log transformation does not allow for ad hoc updates. However, as transactions are added, the log transformation approach requires the necessary information to guarantee correct integration. The log transformation approach is flexible. Since the database values restored after any partition are dependent on the histories of transactions executed during that particular partition, different situations will be reflected by restoring different values.

The details of the log transformations approach can be found in [6] and [7], included in the appendices.

While the log transformation approach was not originally designed to handle complex communications failures such as chronically slow communications or overlapping partitions, it can be modified to work effectively in a network experiencing such failures. Since each site merges logs independently, each site can add transactions to its merge logs as they are available and proceed with its own log transformations regardless of the actions at other sites. Furthermore, as communication with other sites becomes possible, the partition logs from other sites can be readily merged with existing merge logs.

The next section discusses an architecture within which a suitably modified log transformation algorithm might be implemented to allow updates to continue during any kind of communication failure.

3.3 A New Approach

Both data-patch and log transformation were developed to handle the problem of merging divergent database copies after updates were processed during a partition. Under both approaches, there were two distinct modes of operation: a "normal" mode when all sites can communicate with each other, and a "partition" mode. It was assumed that when communication became too slow (according to some application-dependent parameter), sites would switch from normal to partition mode. When normal communications were restored, the divergent copies would be merged. Using this scenario, database copies would always be consistent during normal operation.

In some situations, however, it is more reasonable to operate in a single mode, one that can handle all types of communications problems, even if database copies are

inconsistent for longer periods of time. If communications failures may happen fairly frequently, a site may find itself partitioned from other sites before it has recovered from the previous partition. In particular, situations can arise in which two sites can never exchange information.

Also, the process of merging database states after a partition can degrade system performance, because a site cannot process any new updates until its merge process is completed. This is a problem first because it potentially reduces the amount of time between communications failures during which sites can process new updates, and second because it may simply cause intolerable functional delays. Even the delays caused by concurrency control procedures during normal operation may be reduced by operating in a single, more optimistic, mode. If, for example, it is known that very few transactions at different sites will conflict, or that communications among sites will usually be very slow, it may not be worth incurring the overhead of waiting for locks and for acknowledgments from all sites.

To handle these situations, we have begun extending some of the ideas developed in our work on data-patch and log transformations. The fundamental idea is to divide each site's copy of the database into two parts: an append-only database containing a history of updates known at that site, and a database whose values are derived from this history database, known as a view, that represents the site's best knowledge of the state of the world.

As shown in Figure 3.4, transactions read the local view and send updates to the local distributor. The distributor passes the updates on to the local history database and guarantees that the updates will eventually be delivered to all the other sites. The checker reads the history database, checks any integrity constraints, triggers necessary compensating transactions, and updates the view to reflect relevant changes to the history database.

The correctness of each site's view is tied to that site's knowledge of updates. When the history databases at all sites reflect the same set of updates, the views will be mutually consistent. It is the checker's responsibility to resolve conflicts among transactions. The checker may do this by using techniques from data-patch and from log transformation, as long as the history database includes all necessary information, and so different techniques may be used for the parts of the integration

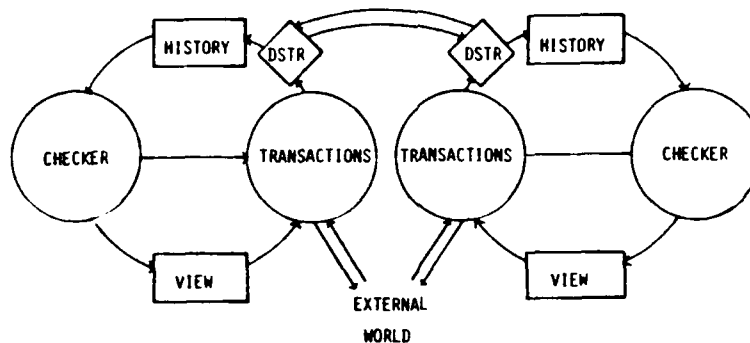


Figure 3.4 Site Architecture

for which they are best suited.

This approach does not ensure a serial order of transactions during times when communication among the sites is good. In the face of communications failures, however, this approach is more general than either data-patch or log transformation. The correctness criteria used in data-patch and log transformation are special cases of the more general idea of view correctness. The old definitions can be built into the definition of the mapping from the history database to the view. Furthermore, knowledge of integrity constraints and compensating actions are embedded in the checker, thus isolating these tests from the transactions.

The new approach is also more flexible. It is designed to handle complex communications failures routinely. The entire history of updates is available, should any extraordinary situations arise. The mechanisms embedded in the checker to map from the history database to the view are versatile; view mapping definitions can be changed without requiring any new techniques. In this way, different integration rules can be used as the application changes and as more sophisticated and efficient integration techniques are developed.

Another advantage of this approach is that the checker assumes all responsibility for testing integrity, invoking any necessary compensating transactions, and updating the view. By isolating these activities, the transaction designer is freed from the burden of dealing with the many types of anomalies that can arise due to complex communications failures. It appears that these anomalies are most easily conceived of and described as undesirable conditions on the database state (essentially as a type of "negative" integrity constraint), and therefore it is more natural and straightforward for the DBA to specify them as such to the checker. The transactions need not be defined to handle all possible anomalous interactions with other transactions. Furthermore, the techniques the checker uses to test for anomalies can be tuned for accuracy and efficiency without affecting transaction definitions or the choice of techniques in checkers at other views. Such flexibility may, however, make this new approach less efficient or may require more work on the part of the DBA.

Our research under the follow-on contract will center on devising powerful, efficient and correct algorithms for implementing the distributor and the checker under this new architecture. We expect the checker algorithms will be extensions of the data patch and log transformation concepts presented here.

4. Project References

This section lists documents related to the development and implementation of the subject contract. Numbered references to these items, enclosed in square brackets ([]) are inserted where applicable in the body of this report.

- [1] Allen, T.A., D.R. Ries, B.T. Blaustein, and R.M. Chilenskas, "Aircraft Recovery Database Requirements and Initial Design," Working Paper, Contract F30602-82-C-0037, Computer Corporation of America, Cambridge, MA (August 1982).
- [2] Allen, T.A., and D.R. Ries, "Aircraft Recovery Transactions and Database Design - Draft (U)," Working Paper, Contract F30602-82-C-0037, Computer Corporation of America, Cambridge, MA (6 April 1983), SECRET.
- [3] Allen, T.A., and D.R. Ries, "Aircraft Recovery Database Design for the SAC C₃ Experiment (U)," Working Paper, Contract F30602-82-C-0037, Computer Corporation of America, Cambridge, MA (October 1983), SECRET.
- [4] Alsberg, P. A. and J. D. Day, "A Principle for Resilient Sharing of Distributed Resources," Proc. of the Second Intl. Conf. on Software Engineering, Pittsburgh, October 1976.
- [5] Barnett, J., C.W. Kaufman, D.R. Ries, B.T. Blaustein, R.M. Chilenskas, and W.K. Lin, "DACOS: A Database Centered Office System," Working Paper, Contract F30602-82-C-0037, Computer Corporation of America, Cambridge, MA (May 1983).
- [6] Blaustein, B.T., R.M. Chilenskas, H. Garcia-Molina, D.R. Ries, and T. Allen, "Partition Recovery Using Semantic Knowledge," Working Paper, Computer Corporation of America, Cambridge, MA (November 1982).
- [7] Blaustein, B., et al., "Maintaining Replicated Databases Even in the Presence of Network Partitions," Proceedings of IEEE, 1983 Electronics and

- Aerospace Conference, Washington, DC (September 1983).
- [8] Chilenskas, R., et al., "Concurrency After the Fact," Proceedings of the 2nd Symposium on Reliability in Distributed Software and Database Systems, Pittsburgh, PA (July 1982).
- [9] Computer Corporation of America, "Business Proposal No.: RSD-82-09-5, Survivable C₃ User Interfaces," Cambridge, MA (October, 1982)
- [10] Garcia-Molina, H., T. Allen, B.T. Blaustein, R.M. Chilenskas, and D.R. Ries, "Data-Patch: Integrating Inconsistent Copies of a Database After a Partition," Proceedings of the 2nd ACM Symposium on Principles of Database Systems, Atlanta, GA (March 1983).
- [11] Gifford, D. "Weighted Voting for Replicated Data," Operating Systems Review, Vol. 13, No. 5, December 1979.
- [12] Kaufman, C.W., J. Barnett, and B.T. Blaustein, "The DACOS Forms Based Query System," Journal of Telecommunications Networks (January 1984).
- [13] Kirkland, M.R., et al., "A Strawman Concept of HERT Operations in the 1990s (U)," Contract F19628-81-C-0001, Project 8020, The MITRE Corporation, Omaha, NE (July 1981), SECRET.
- [14] Kirkland, M.R., J. Mitchell, and T.G. Wiedman, "A Strawman Concept of HERT Operations in the 1990s (U)," Technical Report, MTR-8626, Contract F19628-81-C-0001, Project 8020, The MITRE Corporation, Omaha, NE (Sept. 1981), SECRET.
- [15] Kirkland, M.R., J. Mitchell, and T.G. Wiedman, "Strawman Concept for HERT Operations in a Packet Radio and Distributed Database Environment (U)," Working Paper, WP-23179, The MITRE Corporation, Omaha, NE (1981), SECRET.
- [16] Lin, W.K., and D. R. Ries, "Office Procedures as a Distributed Database Application," Proceedings of SIGMOD 1983, San Jose, CA (May 1983).
- [17] Ries, D.R., W.K. Lin, J. Barnett, and R.M. Chilenskas, "An Architecture for a Database Centered Office System (DACOS)," Journal of

Telecommunications Networks (January 1984).

- [18] Ries, D.R., "Continued and Correct Operation of Distributed Databases in the Presence of Network Partitions," Proceedings of the IEEE COMPSAC 1982 Conference, Chicago, IL (November 1982).
- [19] Rininger, J.B., R.E. Gilligan, T. Allen, and D.R. Ries, "Software Specification for the SAC C₃ Testbed," Special Technical Report 1, Contract F30602-83-C-0027 and F30602-82-C-0037, SRI Project 5453, jointly prepared by SRI International, Menlo Park, CA and Computer Corporation of America, Cambridge, MA (October 1983).
- [20] Skeen, D., "On Network Partitioning," Proc. of the Sixth Berkeley Workshop on Distributed Data Management and Computer Networks, Asilomar, February, 1982.
- [21] Spear, A.C., "Functional System Description for Database Management Application (U)," Working Paper, WP-23197, Contract F19628-84-C-0001, Project 4040, The MITRE Corporation, Omaha, NE (February 1984), SECRET.
- [22] Stonebraker, M., "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES," Proc. of the Third Berkeley Workshop on Distributed Data Management and Computer Networks, San Francisco, August, 1978.
- [23] Thomas, R., "A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases," ACM Transactions on Database Systems, Vol. 4, No. 2, June 1979.
- [24] Wiedman, T.G., "Functional System Description for Recovery Planning Application (U)," Working Paper, WP-23190, Contract F19628-82-C-0001, Project 8020, The MITRE Corporation, Omaha, NE (23 May 1983), SECRET.
- [25] Wiedman, T.G., "Functional System Description for Recovery Planning Application (U)," Working Paper, Rev. 1, WP-23190, Contract F19628-82-C-0001, Project 8020, The MITRE Corporation, Omaha, NE (15 July 1983), SECRET.

- [26] Wiedman, T.G., "Functional System Description for Recovery Planning Application (U)," Working Paper, Rev. 2, WP-23190, Contract F19628-82-C-0001, Project 8020, The MITRE Corporation, Omaha, NE (January 1984), SECRET.
- [27] Wiedman, T.G., "Database Requirements for the Strategic C₃ Experiment (U)," Working Paper, WP-23196, Contract F19628-84-C-0001, Project 4040, The MITRE Corporation, Omaha, NE (February 1984), SECRET.
- [28] Wiedman, T.G., "System Test Plan for Strategic C₃ Experiment - Total System Demonstration," Revision 1, WP-23185, Contract F19628-82-C-0001, Project 8020, The MITRE Corporation, Omaha, NE (May 1982).
- [29] Wiedman, T.G., "System Test Plan for Strategic C₃ Experiment - Total System Demonstration, Revision 2, WP-23185, Contract F19628-82-C-0001, Project 8020, The MITRE Corporation, Omaha, NE (Oct. 1982).
- [30] Wiedman, T.G., "System Test Plan for Strategic C₃ Experiment - Total System Demonstration," Revision 3, WP-23185, Contract F19628-82-C-0001, Project 8020, The MITRE Corporation, Omaha, NE (March 1983).
- [31] Wiedman, T.G., "System Test Plan for Strategic C₃ Experiment - Total System Demonstration," Revision 4, WP-23185, Contract F19628-82-C-0001, Project 8020, The MITRE Corporation, Omaha, NE (Oct. 1983).
- [32] Wiedman, T.G., and A.C. Spear, "System Test Plan for Strategic C₃ Experiment. - Total System Demonstration, Revision 5, WP-23185, Contract F19628-82-C-0001, The MITRE Corporation, Omaha, NE (Feb. 1984).